
A Reproducible Methodology for Benchmarking KV-Caching in Vision-Language Models

Anthony Joseph Ventura
University of Waterloo
aj2ventu@uwaterloo.ca

Philip William Ventura
University of Waterloo
pwventu@uwaterloo.ca

Abstract

Vision-language models have achieved remarkable performance across multimodal tasks, but their practical deployment is constrained by inference efficiency. While key-value (KV) caching has become standard for accelerating autoregressive generation, its isolated impact on multimodal inference remains poorly characterized due to conflation with other optimizations. We address this gap by establishing rigorous protocols for measuring inference optimization effects in isolation and applying them to a PaLI-Gemma-style vision-language transformer. Through controlled experiments at 128, 256, and 512 tokens, we demonstrate that KV-caching provides $10\text{--}13\times$ latency reduction, maintains stable throughput of approximately 10 tokens/second, and counterintuitively reduces peak memory by 8–19%. Beyond these empirical baselines, we contribute a methodological framework based on four principles: confound elimination, phase separation, warm-up exclusion, and statistical rigor. Researchers can apply these to evaluate any inference optimization technique. Our reproducible measurements establish reference baselines for benchmarking future optimizations.

1 Introduction

Modern vision-language models (VLMs) such as PaLI, Flamingo, and PaLI-Gemma have achieved state-of-the-art results across visual reasoning, question-answering, and image generation tasks by combining pretrained vision encoders with large autoregressive language decoders[1–3]. While training costs for these models are substantial, the primary issue when it comes to practical deployment is inference-time efficiency, particularly due to the sequential nature of autoregressive token generation.

Key-value (KV) caching has emerged as the standard solution to accelerate inference in transformer-based systems. By storing and reusing previously computed attention keys and values, KV-caching reduces the per-token computational complexity of self-attention from $O(T^2)$ to $O(T)$ with respect to sequence length T [4]. This optimization is now ubiquitous in production systems including HuggingFace Transformers, vLLM, and TensorRT-LLM.

Despite its widespread adoption, the isolated impact of KV-caching on multimodal inference remains poorly characterized. Existing studies often conflate caching benefits with other optimizations such as fused kernels, quantization, and memory allocators, making it difficult to attribute performance gains to specific mechanisms. This lack of controlled empirical evidence creates challenges for researchers evaluating new inference optimizations and practitioners making deployment decisions.

We address this gap by providing a rigorous, reproducible baseline for measuring KV-caching effects in vision-language models. Using a PaLI-Gemma-style architecture with publicly available pretrained weights (PaLI-Gemma-3B-PT-224-sm) [3], we conduct controlled experiments comparing autoregressive decoding with and without KV-caching enabled. We measure steady-state latency, throughput, and peak memory usage across multiple sequence lengths under identical hardware conditions, implementing strict controls to isolate caching effects from confounding optimizations.

Our contributions are threefold:

1. **Methodological Framework:** We establish rigorous protocols for isolating inference optimization effects, including warm-up handling, prefill/decode separation, and statistical reporting standards that can be applied to study any inference mechanism.
2. **Reference Baselines:** We provide quantitative measurements at 128, 256, and 512 token sequence lengths that researchers can use to benchmark new optimization techniques against established methods.
3. **Reproducible Infrastructure:** We document our experimental setup in detail and encourage the community to extend this benchmark to additional models and configurations, creating a shared resource for inference optimization research.

Importantly, our goal is not to propose novel algorithms but to treat inference mechanisms as first-class subjects of empirical study rather than opaque engineering conveniences. By providing clean, reproducible measurements, we enable more principled comparisons of inference optimizations and more informed deployment decisions for practitioners.

2 Model Architecture

Our baseline model functions as a PaLI-Gemma style VLM, without any fine-tuning or weight modifications, that consists of three key components [3]:

1. **Vision Encoder:** To extract dense image representation, we use the Vision Transformer-based Model (ViT) described by Dosovitskiy et al. [5].
2. **Text Encoder:** To process textual prompts, we use a general transformer-based language model.
3. **Multimodal Fusion and Autoregressive Decoder:** Finally, we use a unified transformer that concatenates text and visual embeddings and performs causal decoding for generation.

This architecture aims to extensively describe images through a generative framework built through CLIP-style contrastive learning and ViT vision encoders.

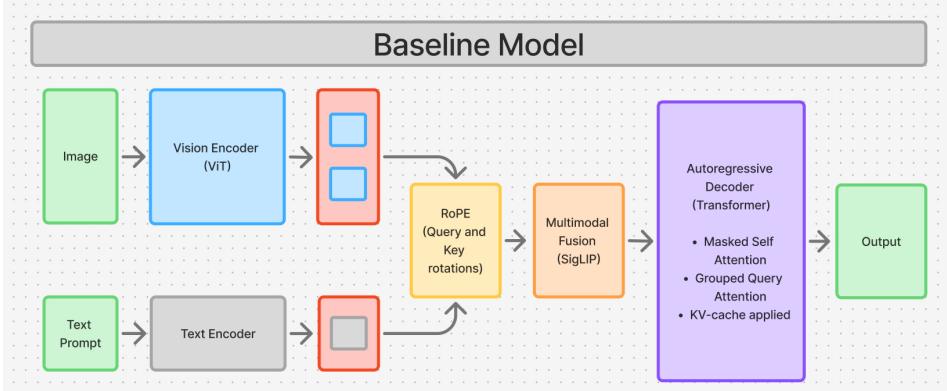


Figure 1: PaLI-Gemma-style VLM architecture used as a baseline model.

More specifically, for the transformer backbone, we chose to adopt the multi-head self-attention mechanism that was standardized by the well-known “Attention Is All You Need” paper [4]. Given input token embeddings $X \in \mathbb{R}^{n \times d}$, the attention output is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where the query, key, and value projections are given by:

$$Q = XW_Q, K = XW_K, V = XW_V$$

As for the decoder stack, it operates in an autoregressive manner, meaning that it masks future positions while computing the attention for the previous and current tokens only, ensuring causal generation.

Furthermore, the baseline model implements three engineered optimizations:

Rotary Positional Embeddings (RoPE)

To begin, we preserve relative positional relations by implementing Rotary Positional Embeddings (RoPE). RoPE uses multiplicative, rotation-based representation, instead of traditional additive positional encodings. More specifically, to insert relative positions in the attention score, RoPE applies a rotation matrix R_p to every single query and key vector based on two token positions i and j :

$$Q_i \leftarrow R_i Q_i, \quad K_j \leftarrow R_j K_j$$

and the attention score becomes a function of relative phases:

$$Q_i K_j^\top = (R_i Q_i)(R_j K_j)^\top = Q_i R_i^\top R_j K_j$$

Encoding this relative positional information efficiently in the dot product, this rotation does not require straightforward positional embeddings. RoPE demonstrates clear advantages, with sequence-length flexibility and decaying inter-token dependencies over distance, which is crucial for stable and scalable training and inference, all being considered desirable properties [6].

Contrastive Vision–Language Pre-training

To combine textual and visual embeddings, we employ a contrastive learning framework inspired by Sigmoid Loss for Language–Image Pre-training (SigLIP). Both the visual and textual encoders are initialized from SigLIP-pretrained weights and project their respective inputs into a shared embedding space, enabling cross-modal alignment.

In traditional Contrastive Language–Image Pre-training (CLIP), training is performed on a batch of N image–text pairs using a symmetric softmax-based contrastive objective. Given normalized image and text embeddings \mathbf{v}_i and \mathbf{t}_i , the image-to-text loss is defined as:

$$\mathcal{L}_{\text{CLIP}}^{\text{img} \rightarrow \text{text}} = -\frac{1}{N} \sum_{i=1}^N \log \frac{\exp(\mathbf{v}_i^\top \mathbf{t}_i / \tau)}{\sum_{j=1}^N \exp(\mathbf{v}_i^\top \mathbf{t}_j / \tau)},$$

with an analogous text-to-image loss computed symmetrically. Here, τ is a learned temperature parameter that scales cosine similarities and controls the sharpness of the alignment distribution [7].

SigLIP distinguishes itself from this formulation by replacing the global softmax normalization with a pairwise sigmoid loss applied independently to each image–text pair. Rather than enforcing competition across all samples in a batch, SigLIP frames alignment as a binary classification task over all possible image–text combinations:

$$\mathcal{L}_{\text{SigLIP}} = -\frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N \log \sigma(y_{ij} (\mathbf{v}_i^\top \mathbf{t}_j \cdot \tau + b)),$$

where $y_{ij} = +1$ if image i and text j form a matching pair and $y_{ij} = -1$ otherwise, $\sigma(\cdot)$ denotes the sigmoid function, and b is a learned bias term.

By eliminating batch-wise normalization, SigLIP reduces sensitivity to batch size and decouples pairwise similarity estimation from global competition. This modification simplifies optimization and has been shown to improve training stability across a range of scaling regimes, while preserving strong visual–textual alignment [8].

Efficient Attention through Grouped Query Attention

To reduce computational load and memory throughout the attention mechanism, we use Grouped Query Attention (GQA). This is a variation of multi-query attention where query heads are grouped while keys and values are shared within specific groups, reducing the number of distinct KV projections and cache size when KV-caching is enabled [9].

In short, our baseline model provides a comprehensive environment to measure the effects of KV-caching. Indeed, the model implements a ViT and textual encoder, with embeddings passing through RoPE, to then be combined by SigLIP. It is finally processed by an autoregressive decoder that employs GQA to maximize efficiency.

3 Methodology

A core contribution of this work is establishing rigorous protocols for measuring inference optimizations in isolation. We identify four critical principles that ensure attribution of performance differences to specific mechanisms rather than confounding factors:

1. **Confound Elimination:** We disable all optimizations except the target mechanism (KV-caching). Specifically, we exclude fused kernels, quantization, tensor parallelism, and speculative decoding. This ensures observed differences arise solely from caching behavior.
2. **Phase Separation:** Inference consists of distinct computational phases with different characteristics. The prefill phase (encoding image and prompt tokens) has identical cost regardless of caching, as no prior context exists to reuse. We therefore isolate and measure only the decode phase, where caching effects manifest.
3. **Warm-up Exclusion:** Initial generation steps exhibit transient behavior as cache structures populate and GPU kernels optimize. We exclude the first 32 tokens from measurements to capture steady-state performance rather than initialization artifacts.
4. **Statistical Rigor:** Single-run measurements are insufficient due to GPU scheduling variance and sampling stochasticity. We conduct multiple independent runs per condition and report means with 95% confidence intervals computed via Student’s t-distribution.

These principles form a reusable framework applicable to evaluating any inference optimization technique. Researchers can adapt this methodology to study FlashAttention, quantization schemes, speculative decoding, or other mechanisms by isolating the target optimization while controlling for confounds.

3.1 Experimental Setup

We evaluate our baseline model in two configurations:

- KV-cache enabled: Standard deployment configuration storing attention keys and values
- KV-cache disabled: Recomputation of full attention context at each decoding step

When KV-caching is enabled, we store key and value matrices from previous layers. Formally, for each layer ℓ , we maintain

$$K_{1:t-1}^\ell, V_{1:t-1}^\ell$$

and only compute new K_t, V_t for token t , reducing attention complexity from $O(T^2)$ to $O(T)$ [4].

Following our confound elimination principle, we disable fused kernels, quantization, tensor parallelism, and speculative decoding. All experiments execute on identical hardware with mixed precision (model default) to ensure fair comparison.

To be precise, all experiments were conducted on an NVIDIA GeForce RTX 2060 GPU (6 GB VRAM) with 16 GB system RAM. Software environment: CUDA 12.8.61, PyTorch 2.8.0+cu128, Python 3.13.7.

3.2 Benchmark dataset and prompts

To ensure realistic multimodal inference behavior, we select five images from MS-COCO 2017 validation set across diverse semantic categories (animals, people, food, landscapes, indoor scenes) [10]. Images are cached locally after initial download to guarantee reproducibility.

We use natural-language prompts designed to produce multi-sentence captions, avoiding trivial early-termination cases and encouraging stable long-sequence decoding behavior representative of production workloads.

3.3 Generation Protocol

Following our warm-up exclusion principle, we treat the first 32 tokens as warm-up to allow cache structure population and GPU kernel optimization. We measure only subsequent tokens to capture steady-state performance.

Following our phase separation principle, we execute prefill (encoding image and prompt tokens) prior to measurement but exclude it from timing, as prefill cost is independent of caching. We benchmark only the autoregressive decode phase.

To ensure statistical rigor, we conduct 150 total runs: 5 independent runs per image, per sequence length (128, 256, 512 tokens), per condition (cached/uncached). We use top-p sampling with fixed hyperparameters across all runs.

For each condition, we measure:

- Steady-state per-token latency (ms/token)
- Throughput (tokens/second)
- Peak GPU memory (CUDA allocation during decode only)
- Token count

We report means, 95% confidence intervals, and standard deviations for all metrics.

3.4 Implementation Details

We reset CUDA peak memory statistics before decoding to isolate cache memory effects. Additional implementation controls include manual KV-cache object management, patched position-ID handling to prevent overflow, patched rotary embedding forward pass, separation of image/text embeddings in multimodal fusion, and consistent sampling policies. We reuse random seeds where applicable but preserve sampling stochasticity, as we study runtime performance rather than output variance.

3.5 Experimental Objective

Following standard practice for inference benchmarking, we do not evaluate caption quality or semantic accuracy. All generated text is discarded except for token counts. Our sole objective is quantifying computational efficiency under matched conditions.

This design ensures the only systematic difference between configurations is the presence or absence of KV-caching. Observed performance differences can therefore be attributed directly to the caching mechanism itself.

4 Results

4.1 Baseline Performance Measurements

We present reference measurements characterizing KV-caching effects across three sequence lengths in a controlled multimodal transformer setting.

Table 1 summarizes our measurements across output sequence lengths of 128, 256, and 512 tokens under both configurations. The following subsections detail specific performance characteristics that emerge from our controlled comparison.

Table 1: Inference results across sequence lengths with and without KV caching. Reported are means with 95% confidence intervals (CI) and standard deviations (SD).

Seq. Len	KV Cache	Samples	Steady-State TPS	Ms/Token	Peak Memory (MB)	Tokens Gen.
128	Enabled	25	10.20 (CI 0.05; SD 0.11)	98.08 (CI 0.44; SD 1.05)	6547.58 (CI 1.27; SD 3.00)	128.0
	Disabled	25	1.02 (CI 0.01; SD 0.02)	984.86 (CI 8.99; SD 21.33)	7122.91 (CI 1.58; SD 3.75)	128.0
256	Enabled	25	10.17 (CI 0.01; SD 0.03)	98.35 (CI 0.12; SD 0.28)	6547.58 (CI 1.27; SD 3.00)	256.0
	Disabled	25	0.92 (CI 0.00; SD 0.01)	1082.63 (CI 2.96; SD 7.03)	7437.55 (CI 1.50; SD 3.56)	256.0
512	Enabled	25	10.46 (CI 0.29; SD 0.68)	95.98 (CI 2.35; SD 5.57)	6547.58 (CI 1.27; SD 3.00)	512.0
	Disabled	25	0.79 (CI 0.01; SD 0.03)	1268.43 (CI 20.55; SD 48.79)	8069.92 (CI 1.66; SD 3.93)	512.0

4.2 Decoding Latency

Enabling KV-caching produces substantial latency reductions across all sequence lengths. At 128 tokens, average per-token latency decreases from 984.86 ms to 98.08 ms ($10.0\times$ speedup). At 512 tokens, latency decreases from 1268.43 ms to 95.98 ms ($13.2\times$ speedup).

Figure 2 illustrates the scaling behavior, where latency without caching increases rapidly with sequence length (quadratic growth), while latency with caching remains approximately constant (linear growth). These measurements indicate that KV-caching provides order-of-magnitude latency improvements in production deployment. Also, the benefits scale with sequence length, making caching increasingly critical for longer generations.

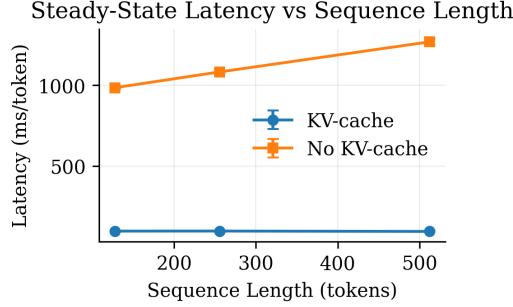


Figure 2: Decoding latency vs. sequence length for KV-cache enabled and disabled configurations.

4.3 Throughput

Generation throughput exhibits similar patterns. Without caching, throughput degrades from 1.02 tokens/s at 128 tokens to 0.79 tokens/s at 512 tokens. With caching enabled, throughput remains stable at approximately 10 tokens/s across all sequence lengths (Figure 3).

The throughput stability with caching, that remains nearly independent of sequence length, contrasts with the degradation observed without caching. This confirms that uncached decoding’s dominant cost is repeated full-context recomputation at each step. Therefore, stable throughput across sequence lengths simplifies capacity planning and resource allocation for production systems. Without caching, longer sequences impose nonlinear computational penalties that complicate deployment.

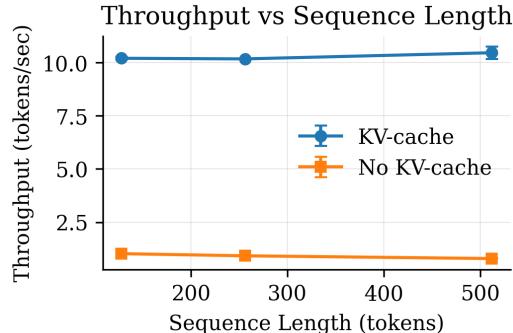


Figure 3: Throughput vs. sequence length for KV-cache enabled and disabled configurations.

4.4 Speedup Scaling

Figure 4 reports relative speedup factors: $10.0\times$ at 128 tokens, $11.0\times$ at 256 tokens, and $13.2\times$ at 512 tokens. The increasing speedup with sequence length aligns with theoretical predictions: as T grows, the gap between $O(T^2)$ uncached and $O(T)$ cached complexity widens proportionally [4]. This scaling pattern provides a baseline for evaluating alternative optimizations. Thus, novel inference

techniques should be compared against these established speedup curves to assess whether they provide complementary or superior benefits.

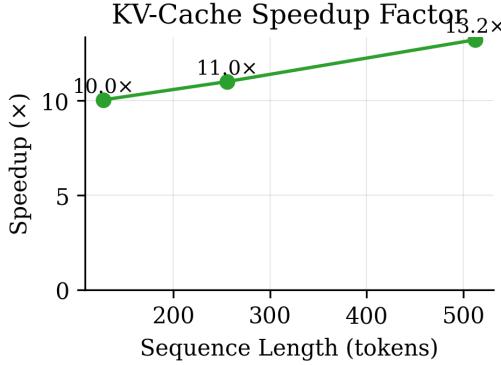


Figure 4: KV-cache speedup factor relative to uncached decoding across sequence lengths.

4.5 Memory Efficiency

Peak GPU memory consumption presents a counterintuitive result: disabling KV-caching increases memory usage despite performing less work per token. Memory rises from 6547.58 MB (cached) to 7122.91 MB, 7437.55 MB, and 8069.92 MB (uncached) at 128, 256, and 512 tokens respectively (Figure 5).

This occurs because uncached decoding repeatedly allocates and recomputes intermediate activations for the entire sequence prefix at each step, whereas KV-caching stores only compact key-value tensors. Thus, caching improves both computational efficiency and memory efficiency, a dual benefit critical for resource-constrained deployments. That said, KV-caching provides memory savings in addition to speed improvements, contradicting the common assumption that caching trades memory for speed. This makes caching beneficial even in memory-limited scenarios.

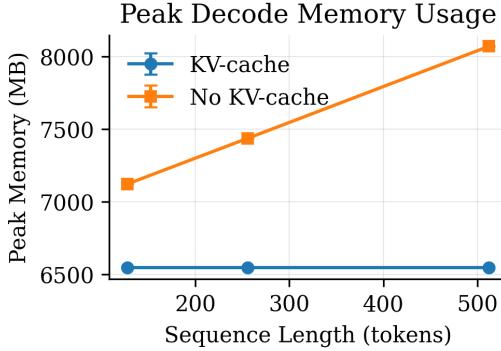


Figure 5: Peak GPU memory usage for cached and uncached decoding across sequence lengths.

4.6 Complexity Verification

Figure 6 presents log-log scaling analysis, where the uncached decoding curve exhibits a slope consistent with quadratic scaling, while the KV-cached curve is approximately linear. This visualization confirms that KV-caching fundamentally alters the computational scaling regime rather than providing a constant factor optimization. For researchers, the clear separation in scaling behavior validates that our measurements capture the theoretically predicted complexity transition.

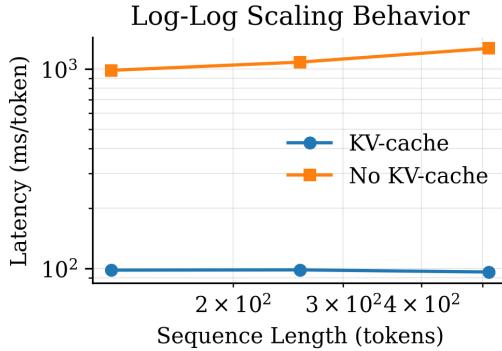


Figure 6: Log–log scaling of decoding time showing quadratic growth without KV-caching and near-linear growth with KV-caching.

4.7 Summary of Measurements

Our controlled comparison establishes the following baseline characteristics for KV-caching in vision-language transformers:

- Latency reduction: $10\text{--}13\times$ speedup, increasing with sequence length
- Throughput stability: ≈ 10 tokens/s maintained across sequence lengths (vs. < 2 tokens/s degrading without cache)
- Memory efficiency: 8–19% memory reduction despite performing more efficient computation
- Scaling regime: Fundamental shift from $O(T^2)$ to $O(T)$ complexity

These measurements were obtained under strict experimental controls (Section 3), with no model modifications or auxiliary optimizations. Performance differences are attributable solely to the caching mechanism itself.

In terms of application, researchers can use these baselines when evaluating new inference optimizations. On the other hand, practitioners can use these measurements to estimate expected production performance and inform deployment decisions.

Limitations include baselines that are specific to our hardware configuration (Section 3), model architecture (Section 2), and workload characteristics (natural image captioning). Performance on other hardware, architectures, or workloads may differ.

5 Conclusion

5.1 Summary

We present a rigorous methodology for isolating inference optimization effects in vision-language models, using KV-caching as a case study. By controlling all variables except the cache mechanism, we demonstrate how to attribute performance differences to specific optimizations. Our baseline measurements show KV-caching provides $10\text{--}13\times$ speedup, maintains ≈ 10 tokens/s throughput, reduces memory by 8–19%, and shifts computational scaling from $O(T^2)$ to $O(T)$.

Beyond these empirical results, we establish four principles for rigorous inference studies: confound elimination, phase separation, warm-up exclusion, and statistical rigor. This framework is immediately applicable to evaluating FlashAttention [11], quantization [12], speculative decoding [13], or any other optimization technique.

5.2 Extending This Work

We encourage an extension of these measurements to larger models (7B, 13B+ parameters), diverse hardware platforms (consumer GPUs, TPUs, data center accelerators), longer sequences (1K–8K+ to

kens), and different workloads (VQA, dense captioning, reasoning tasks). Additionally, investigating interactions between KV-caching and other optimizations, such as quantization [12], FlashAttention [11], or speculative decoding [13], would reveal how these techniques complement or interfere with each other in production systems.

Our baseline measurements also reveal open research questions: At what batch sizes does cache overhead outweigh computational savings? How do emerging sparse attention mechanisms [14] and linearized approximations [15] interact with caching? Can adaptive eviction policies [16] optimize the size-performance trade-off under real-world deployment constraints? These questions represent promising directions for future work.

5.3 Closing Perspective

As vision-language models scale, inference efficiency increasingly determines practical feasibility. KV-caching is ubiquitous precisely because it provides dramatic, reliable improvements. Our measurements confirm this empirically while establishing a methodology for evaluating future optimizations with similar rigor. By treating inference mechanisms as subjects worthy of careful study, we enable principled design of efficient multimodal systems.

References

- [1] X. Chen, X. Wang, S. Changpinyo, A. J. Piergiovanni, P. Padlewski, D. Salz, S. Goodman, A. Grycner, B. Mustafa, L. Beyer, A. Kolesnikov, N. Ding, K. Rong, H. Akbari, G. Mishra, et al. Pali: A jointly-scaled multilingual language-image model. In *International Conference on Learning Representations (ICLR)*, 2023.
- [2] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.
- [3] A. Steiner, A. Susano Pinto, M. Tschannen, D. Keysers, X. Wang, Y. Bitton, A. Gritsenko, M. Minderer, A. Sherbondy, S. Long, S. Qin, R. Ingle, E. Bugliarello, S. Kazemzadeh, T. Mesnard, et al. Paligemma 2: A family of versatile vlms for transfer. *arXiv preprint arXiv:2412.03555*, 2024.
- [4] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2021.
- [6] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [7] A. Radford, J. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.
- [8] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language-image pre-training (siglip). *arXiv preprint arXiv:2303.15343*, 2023.
- [9] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghi. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- [10] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.
- [11] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022.
- [12] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *arXiv preprint arXiv:2206.01861*, 2022.
- [13] A. Li, J. Gu, and M. Zhang. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*, 2022.
- [14] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [15] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. *arXiv preprint arXiv:2006.16236*, 2020.
- [16] J. Rae, A. B. Ainslie, S. Chiu, et al. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2020.